SIAM J. SCI. COMPUT.
Vol. 38, No. 5, pp. S25–S47

AUTOMATED GENERATION AND SYMBOLIC MANIPULATION OF TENSOR PRODUCT FINITE ELEMENTS*

A. T. T. MCRAE†, G.-T. BERCEA‡, L. MITCHELL§, D. A. HAM§, AND C. J. COTTER¶

Abstract. We describe and implement a symbolic algebra for scalar and vector-valued finite elements, enabling the computer generation of elements with tensor product structure on quadrilateral, hexahedral, and triangular prismatic cells. The algebra is implemented as an extension to the domain-specific language UFL, the Unified Form Language. This allows users to construct many finite element spaces beyond those supported by existing software packages. We have made corresponding extensions to FIAT, the FInite element Automatic Tabulator, to enable numerical tabulation of such spaces. This tabulation is consequently used during the automatic generation of low-level code that carries out local assembly operations, within the wider context of solving finite element problems posed over such function spaces. We have done this work within the code-generation pipeline of the software package Firedrake; we make use of the full Firedrake package to present numerical examples.

Key words. automated code generation, tensor product finite element, finite element exterior calculus

AMS subject classifications. 65M60, 65N30, 68N20

DOI. 10.1137/15M1021167

1. Introduction.

Many different areas of science benefit from the ability to generate approximate numerical solutions to PDEs. In the past decade, there has been increasing use of software packages and libraries that automate fundamental operations. The FEniCS Project [27] is especially notable for allowing the user to express discretizations of PDEs, based on the finite element method, in UFL (the Unified Form Language) [4, 2], a concise, high-level language embedded in Python. Corresponding efficient low-level code is automatically generated by FFC (the FEniCS Form Compiler) [24, 28], making use of FIAT (the FInite element Automatic Tabulator) [22, 23]. These local "kernels" are executed on each cell[1] in the mesh, and the resulting global systems of equations can be solved using a number of third-party libraries.

There are multiple advantages to having the discretization represented symbolically within a high-level language. The user can write down complicated expressions concisely without being encumbered by low-level implementation details. Suitable optimizations can then be applied automatically during the generation of low-level code; this would be a tedious process to replicate by hand on each new expression. Such

*Received by the editors May 13, 2015; accepted for publication (in revised form) March 24, 2016; published electronically October 27, 2016. This work was supported by the Grantham Institute and Climate-KIC, the Natural Environment Research Council [grants NE/K006789/1, NE/K008951/1, and NE/M013480/1], and an Engineering and Physical Sciences Research Council prize studentship.
http://www.siam.org/journals/sisc/38-5/M102116.html

†The Grantham Institute and Department of Mathematics, Imperial College London, London, SW7 2AZ, UK, and Department of Mathematical Sciences, University of Bath, Bath, BA2 7AY, UK (a.t.t.mcrae@bath.ac.uk).

‡Department of Computing, Imperial College London, London, SW7 2AZ, UK (gheorghe-teodor.bercea08@imperial.ac.uk).

§Department of Computing and Department of Mathematics, Imperial College London, London, SW7 2AZ, UK (lawrence.mitchell@imperial.ac.uk, david.ham@imperial.ac.uk).

¶Department of Mathematics, Imperial College London, London, SW7 2AZ, UK (colin.cotter@imperial.ac.uk).

[1]Note on terminology: Throughout this paper, we use the term "cell" to denote the geometric component of the mesh; we reserve the term "finite element" to denote the space of functions on a cell and supplementary information related to global continuity.

transformations have previously been implemented in FFC [34, 24]. In this paper, we extend this high-level approach by introducing a user-facing symbolic representation of tensor product finite elements. First, this enables the construction of a wide range of finite element spaces, particularly scalar- and vector-valued identifications of finite element differential forms. Second, while we have not done this at present, the symbolic representation of a tensor product finite element may be exploited to automatically generate optimal-complexity algorithms via a sum-factorization approach.

Firedrake is an alternative software package to FEniCS which presents a similar—in many cases, identical—interface. Like FEniCS, Firedrake automatically generates low-level C kernels from high-level UFL expressions. However, the execution of these kernels over the mesh is performed in a fundamentally different way; this led to significant performance increases, relative to FEniCS 1.5, across a range of problems [36]. As well as the high-level representation of finite element operations embedded in Python, Firedrake and FEniCS have other attractive features. They support a wide range of arbitrary-order finite element families, which are of use to numerical analysts proposing novel discretizations of PDEs. They also make use of third-party libraries, notably PETSc [10], exposing a wide range of solvers and preconditioners for efficient solution of linear systems.

A limitation of Firedrake and FEniCS has been the lack of support for anything other than fully unstructured meshes with simplicial cells: intervals, triangles, or tetrahedra. There are good reasons why a user may wish to use a mesh of non-simplicial cells. Our main motivation is geophysical simulations, which are governed by highly anisotropic equations in which gravity plays an important role. In addition, they often require high aspect-ratio domains: the vertical height of the domain may be several orders of magnitude smaller than the horizontal width. These domains admit a decomposition which has an unstructured horizontal "base mesh" but with regular vertical layers; we will refer to this as an *extruded* mesh. The cells in such a mesh are not simplices but instead have a product structure. In two dimensions (2D) this leads to quadrilateral cells; in three dimensions (3D), triangular prisms or hexahedra. From a mathematical viewpoint, the vertical alignment of cells minimizes difficulties associated with the anisotropy of the governing equations. From a computational viewpoint, the vertical structure can be exploited to improve performance compared to a fully unstructured mesh.

On such cells, we will focus on producing finite elements that can be expressed as (sums of) products of existing finite elements. This covers many, though not all, of the common finite element spaces on product cells. We pay special attention to element families relevant to finite element exterior calculus, a mathematical framework that leads to stable mixed finite element discretizations of PDEs [7, 8, 6]. This paper therefore describes some of the extensions to the Firedrake code-generation pipeline to enable the solution of finite element problems on cells which are products of simplices. These enable the automated generation of low-level kernels representing finite element operations on such cells. We remark that, due to our geophysical motivations, Firedrake has complete support for extruded meshes whose unstructured base mesh is built from simplices or quadrilaterals. At the time of writing, however, it does not support fully unstructured prismatic or hexahedral meshes.

Many, though not all, of the finite elements we can now construct already have implementations in other finite element libraries. deal.II [11] contains both scalar-valued tensor product finite elements and the vector-valued Raviart–Thomas and Nédélec elements of the first kind [39, 32], which can be constructed using tensor products. However, deal.II only supports quadrilateral and hexahedral cells and has

no support for simplices or triangular prisms. DUNE PDELab [12] contains low-order Raviart–Thomas elements on quadrilaterals and hexahedra but only supports scalar-valued elements on triangular prisms. Nektar++ [16] uses tensor-product elements extensively and supports a wide range of geometric cells but is restricted to scalar-valued finite elements. MFEM [1] supports Raviart–Thomas and Nédélec elements of the first kind, though it has no support for triangular prisms. NGSolve [42, 43] contains many, possibly all, of the exterior-calculus-inspired tensor-product elements that we can create on triangular prisms and hexahedra. However, it does not support elements such as the Nédélec element of the second kind [33] on these cells, which do not fit into the exterior calculus framework.

This paper is structured as follows: In section 2, we provide the mathematical details of product finite elements. In section 3, we describe the software extensions that allow such elements to be represented and numerically tabulated. In section 4, we present numerical experiments that make use of these elements, within Firedrake. Finally, in section 5 and section 6, we give some limitations of our implementation and other closing remarks.

1.1. Summary of contributions.

- The description and implementation of a symbolic algebra on existing scalar- and vector-valued finite elements. This allows for the creation of scalar-valued continuous and discontinuous tensor-product elements, and vector-valued curl- and div-conforming tensor product elements in 2D and 3D.
- Certain vector-valued finite elements on quadrilaterals, triangular prisms, and hexahedra are completely unavailable in other major packages, and some elements we create have no previously published implementation.
- The tensor-product element structure is captured symbolically at runtime. Although we do not take advantage of this at present, this could later be exploited to automate the generation of low-complexity algorithms through sum-factorization and similar techniques.

2. Mathematical preliminaries.

This section is structured as follows: In subsection 2.1, we give the definition of a finite element that we work with. In subsection 2.2, we briefly define the sum of finite elements. In subsection 2.3, we discuss finite element spaces in terms of their intercell continuity. In subsection 2.4 and subsection 2.5, which form the main part of this section, we define the product of finite elements and state how these products can be manipulated and combined to produce elements compatible with finite element exterior calculus. Up to this point, our exposition uses the language of scalar and vector fields as our existing software infrastructure uses scalars and vectors, and we believe this makes the paper accessible to a wider audience. However, we end this section with subsection 2.6, which briefly restates subsection 2.4 and subsection 2.5 in terms of differential forms. These provide a far more natural setting for the underlying operations.

2.1. Definition of a finite element.

We will follow Ciarlet [18] in defining a *finite element* to be a triple (K, P, N) where

- K is a bounded domain in $\mathbb{R}^n$, to be interpreted as a generic *reference cell* on which all calculations are performed,
- P is a finite-dimensional space of continuous functions on K, typically some subspace of polynomials, and
- $N = \{n_1, \ldots, n_{\dim P}\}$ is a basis for the dual space P'—the space of linear functionals on P—where the elements of the set N are called *nodes*.

Let Ω be a compact domain which is decomposed into a finite number of non-overlapping cells. Assume that we wish to find an approximate solution to some PDE, posed in Ω, using the finite element method. A *finite element* together with a given decomposition of Ω produces a *finite element space*.

A finite element space is a finite-dimensional function space on Ω. There are essentially two things that need to be specified to characterize a finite element space: the manner in which a function may vary within a single cell, and the amount of continuity a function must have between neighboring cells.

The former is related to P; more details are given in subsection 2.3.2. A basis for P is therefore very useful in implementations of the finite element method. Often, this is a *nodal basis* in which each of the basis functions $\Phi_1, \ldots, \Phi_{\dim P}$ vanish when acted on by all but one node:

$$n_i(\Phi_j) = \delta_{ij}. \tag{1}$$

Basis functions from different cells can be combined into basis functions for the finite element space on Ω. The intercell continuity of these basis functions is related to the choice of nodes, N. This is the core topic of subsection 2.3.

2.2. Sum of finite elements. Suppose we have finite elements $U = (K, P_A, N_A)$ and $V = (K, P_B, N_B)$, which are defined over the same reference cell K. If the intersection of P_A and P_B is trivial, we can define the *direct sum* $U \oplus V$ to be the finite element (K, P, N), where

$$P := P_A \oplus P_B \equiv \{f_A + f_B \mid f_A \in P_A, f_B \in P_B\}, \tag{2}$$

$$N := N_A \cup N_B. \tag{3}$$

2.3. Sobolev spaces, intercell continuity, and Piola transforms. Finite element spaces are a finite-dimensional subspace of some larger Sobolev space, depending on the degree of continuity of functions between neighboring cells. We will consider finite element spaces in H^1, $H(\text{curl})$, $H(\text{div})$, and L^2.

A brief remark: it is clear that these Sobolev spaces have some trivial inclusion relations; H^1 is a subspace of L^2, $H(\text{div})$ and $H(\text{curl})$ are both subspaces of $[L^2]^d$, where d is the spatial dimension, and $[H^1]^d$ is a subspace of both $H(\text{div})$ and $H(\text{curl})$. However, in what follows, when we make casual statements such as $V \subset H(\text{div})$, it is *implied* that $V \not\subset [H^1]^d$; i.e., we have made the strongest statement possible. In particular, we will use L^2 to denote a total absence of continuity between cells.

2.3.1. Geometric decomposition of nodes. The set of nodes N, from the definition in subsection 2.1, is used to enforce the continuity requirements on the "global" finite element space. This is done by associating nodes with *topological* entities of K—vertices, facets, and so on. When multiple cells in Ω share a topological entity, the cells must agree on the value of any degree of freedom associated with that entity. This leads to coupling between any cells that share the entity. The association of nodes with topological entities is crucial in determining the continuity of finite element spaces; this is sometimes called the *geometric decomposition* of nodes.

For H^1 elements, functions are fully continuous between cells and must therefore be single-valued on vertices, edges, and facets. Nodes are first associated with *vertices*. If necessary, additional nodes are associated with *edges*, then with *facets*, and then with the *interior* of the reference cell.

For $H(\text{curl})$ elements, which are intrinsically vector-valued, functions must have continuous tangential component between cells. The component(s) of the function

tangential to edges and facets must therefore be single-valued. Nodes are first associated with *edges* until the tangential component is specified uniquely. If necessary, additional nodes are associated with *facets*, and then with the *interior* of the reference cell.

For $H(\mathrm{div})$ elements, which are also intrinsically vector-valued, functions must have continuous normal component between cells. The component of the function normal to facets must therefore be single-valued. Nodes are first associated with *facets*. If necessary, additional nodes are associated with the *interior* of the cell.

L^2 elements have no continuity requirements. Typically, all nodes are associated with the *interior* of the cell; this does not lead to any continuity constraints.

2.3.2. Piola transforms. For functions to have the desired continuity on the global mesh, they may need to undergo an appropriate *mapping* from reference to physical space. Let $\vec{X}$ represent coordinates on the reference cell and $\vec{x}$ represent coordinates on the physical cell; for each physical cell there is some map $\vec{x} = g(\vec{X})$.

For H^1 or L^2 functions, no explicit mapping is needed. Let $\hat{f}(\vec{X})$ be a function defined over the reference cell. The corresponding function $f(\vec{x})$ defined over the physical cell is then

$$f(\vec{x}) = \hat{f} \circ g^{-1}(\vec{x}). \tag{4}$$

We will refer to this as the *identity* mapping.

However, if we wish to have continuity of the normal or tangential component of the vector field in physical space, (4) does not suffice. $H(\mathrm{div})$ and $H(\mathrm{curl})$ elements therefore use *Piola transforms* to map functions from reference space to physical space. We will use J to denote $Dg(\vec{X})$, the Jacobian of the coordinate transformation. $H(\mathrm{div})$ functions are mapped using the contravariant Piola transform, which preserves normal components,

$$\vec{f}(\vec{x}) = \frac{1}{\det J} J \hat{\vec{f}} \circ g^{-1}(\vec{x}), \tag{5}$$

while $H(\mathrm{curl})$ functions are mapped using the covariant Piola transform, which preserves tangential components,

$$\vec{f}(\vec{x}) = J^{-T} \hat{\vec{f}} \circ g^{-1}(\vec{x}). \tag{6}$$

2.4. Product finite elements. In this section, we discuss how to take the product of a pair of finite elements and how this product element may be manipulated to give different types of intercell continuity. We will label our constituent elements U and V, where $U := (K_A, P_A, N_A)$ and $V := (K_B, P_B, N_B)$, following the notation of subsection 2.1. We begin with the definition of the product reference cell, which is straightforward. However, the spaces of functions and the associated nodes are intimately related; hence the discussion of these is interleaved.

2.4.1. Product cells. Given reference cells $K_A \subset \mathbb{R}^n$ and $K_B \subset \mathbb{R}^m$, the reference product cell $K_A \times K_B$ can be defined straightforwardly as follows:

$$K_A \times K_B := \left\{ (x_1, \ldots, x_{n+m}) \in \mathbb{R}^{n+m} \mid (x_1, \ldots, x_n) \in K_A, (x_{n+1}, \ldots, x_{n+m}) \in K_B \right\}. \tag{7}$$

The topological entities of $K_A \times K_B$ correspond to products of topological entities of K_A and K_B. If we label the entities of a reference cell (in $\mathbb{R}^n$, say) by their dimension, so that 0 corresponds to vertices, 1 to edges, ..., $n-1$ to facets and n to the cell, the entities of $K_A \times K_B$ can be labeled as follows:

(0, 0): vertices of $K_A \times K_B$—the product of a vertex of K_A with a vertex of K_B
(1, 0): edges of $K_A \times K_B$—the product of an edge of K_A with a vertex of K_B
(0, 1): edges of $K_A \times K_B$—the product of a vertex of K_A with an edge of K_B
$\vdots$
(n-1, m): facets of $K_A \times K_B$—the product of a facet of K_A with the cell of K_B
(n, m-1): facets of $K_A \times K_B$—the product of the cell of K_A with a facet of K_B
(n, m): cell of $K_A \times K_B$—the product of the cell of K_A with the cell of K_B
It is important to distinguish between different types of entities, even those with the same dimension. For example, if K_A is a triangle and K_B is an interval, the $(2,0)$ facets of the prism $K_A \times K_B$ are triangles while the $(1,1)$ facets are quadrilaterals.

2.4.2. Product spaces of functions: Simple elements. Given spaces of functions P_A and P_B, the product space $P_A \otimes P_B$ can be defined as the span of products of functions in P_A and P_B:

$$P_A \otimes P_B := \operatorname{span}\{f \cdot g \mid f \in P_A, g \in P_B\}, \tag{8}$$

where the product function $f \cdot g$ is defined so that

$$(f \cdot g)(x_1, \ldots, x_{n+m}) = f(x_1, \ldots, x_n) \cdot g(x_{n+1}, \ldots, x_{n+m}). \tag{9}$$

In the cases we consider explicitly, at least one of f or g will be scalar-valued, so the product on the right-hand side of (9) is unambiguous. A basis for $P_A \otimes P_B$ can be constructed from bases for P_A and P_B. If P_A and P_B have nodal bases

$$\left\{\Phi_1^{(A)}, \Phi_2^{(A)}, \ldots \Phi_N^{(A)}\right\}, \quad \left\{\Phi_1^{(B)}, \Phi_2^{(B)}, \ldots \Phi_M^{(B)}\right\}, \tag{10}$$

respectively, a nodal basis for $P_A \otimes P_B$ is given by

$$\{\Phi_{i,j}, \quad i = 1, \ldots, N,\, j = 1, \ldots, M\}, \tag{11}$$

where

$$\Phi_{i,j} := \Phi_i^{(A)} \cdot \Phi_j^{(B)}, \quad i = 1, \ldots, N,\, j = 1, \ldots, M; \tag{12}$$

the right-hand side uses the same product as (9).

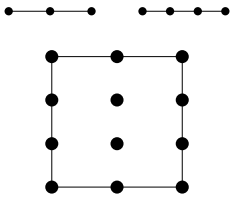
While this already gives plenty of flexibility, there are cases in which a different, more natural, space can be built by further manipulation of $P_A \otimes P_B$. We will return to this after a brief description of product nodes.

2.4.3. Product nodes: Geometric decomposition. Recall that the nodes are a basis for the dual space $(P_A \otimes P_B)'$, and that the intercell continuity of the finite element space is related to the association of nodes with topological entities of the reference cell.

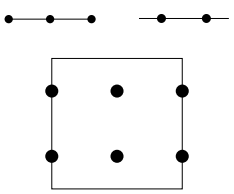
Assuming that we know bases for P_A' and P_B', there is a natural basis for $(P_A \otimes P_B)'$ which is essentially an outer (tensor) product of the bases for P_A' and P_B'. Let $n_{i,j}$ denote a "product" of $n_i^{(A)}$, the ith node in N_A, with $n_j^{(B)}$, the jth node in N_B—typically the evaluation of some component of the function. If $n_i^{(A)}$ is associated with an entity of K_A of dimension p and $n_j^{(B)}$ is associated with an entity of K_B of dimension q, then $n_{i,j}$ is associated with an entity of $K_A \times K_B$ with label (p, q).

This geometric decomposition of nodes in the product element is used to motivate further manipulation of $P_A \otimes P_B$ to produce a more natural space of functions, particularly in the case of vector-valued elements.

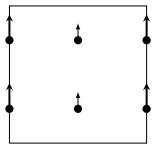
2.4.4. Product spaces of functions: Scalar- and vector-valued elements in 2D and 3D. In 2D, we take the reference cells K_A and K_B to be intervals, so the product cell $K_A \times K_B$ is 2D. Finite elements on intervals are scalar-valued and are either in H^1 or in L^2. We will consider the creation of 2D elements in H^1, $H(\mathrm{curl})$, $H(\mathrm{div})$, and L^2. A summary of the following is given in Table 1.



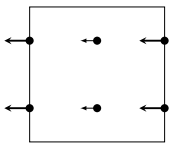
H^1: The element must have nodes associated with vertices of the reference product cell. The vertices of the reference product cell are formed by taking the product of vertices on the intervals. The constituent elements must therefore have nodes associated with vertices and so must both be in H^1.



$H(\mathrm{curl})$: The element must have nodes associated with edges of the reference product cell. The edges of the reference product cell are formed by taking the product of an interval's vertex with an interval's interior. One of the constituent elements must therefore have nodes associated with vertices, while the other must only have nodes associated with the interior. Taking the product of an H^1 element with an L^2 element gives a scalar-valued element with nodes on the $(0,1)$ facets, for example.

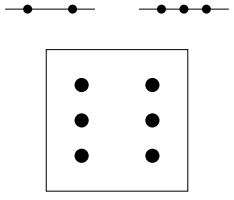


To create an $H(\mathrm{curl})$ element, we now multiply this scalar-valued element by the vector $(0,1)$ to create a vector-valued finite element (if we had taken the product of an L^2 element with an H^1 element, we would multiply by $(1,0)$). This gives an element whose tangential component is continuous across *all* edges (trivially so on two of the edges). In addition, we must use an appropriate Piola transform when mapping from reference space into physical space.



$H(\mathrm{div})$: We create a scalar-valued element in the same way as in the $H(\mathrm{curl})$ case, but multiplied by the "other" basis vector (for $H^1 \times L^2$, we choose $(-1,0)$; the minus sign is useful for orientation consistency in unstructured quadrilateral meshes; for $L^2 \times H^1$, $(0,1)$). This gives an element whose normal component is continuous across *all* edges, and again, we must use an appropriate Piola transform when mapping from reference space into physical space.

Note that the scalar-valued product elements we produce above are perfectly legitimate finite elements, and it is not compulsory to form vector-valued elements from them. Indeed, we use such a scalar-valued element for the example in subsection 4.2. However, the vector-valued elements are generally more useful and fit naturally within Finite Element Exterior Calculus, as we will see in subsection 2.5.



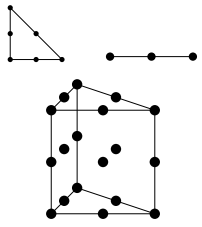
L^2: The element must only have nodes associated with the interior of the reference product cell. The constituent elements must therefore only have nodes associated with their interiors and so must both be in L^2.

TABLE 1
Summary of 2D product elements.

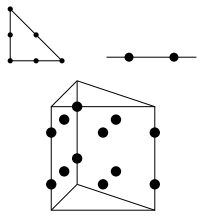
Product (1D × 1D)	Components	Modifier	Result	Mapping
$H^1 \times H^1$	$f \times g$	(none)	fg	identity
$H^1 \times L^2$	$f \times g$	(none)	fg	identity
$H^1 \times L^2$	$f \times g$	$H(\text{curl})$	$(0, fg)$	covariant Piola
$H^1 \times L^2$	$f \times g$	$H(\text{div})$	$(-fg, 0)$	contravariant Piola
$L^2 \times H^1$	$f \times g$	(none)	fg	identity
$L^2 \times H^1$	$f \times g$	$H(\text{curl})$	$(fg, 0)$	covariant Piola
$L^2 \times H^1$	$f \times g$	$H(\text{div})$	$(0, fg)$	contravariant Piola
$L^2 \times L^2$	$f \times g$	(none)	fg	identity

In 3D, we take $K_A \subset \mathbb{R}^2$ and K_B to be an interval, so the product cell $K_A \times K_B$ is 3D. Finite elements on a 2D reference cell may be in H^1, $H(\text{curl})$, $H(\text{div})$, or L^2. Elements on a one-dimensional (1D) reference cell may be in H^1 or L^2. We will consider the creation of 3D elements in H^1, $H(\text{curl})$, $H(\text{div})$, and L^2. A summary of the following is given in Table 2.

Note: In the following pictures, we have taken the 2D cell to be a triangle. However, the discussion is equally valid for quadrilaterals.

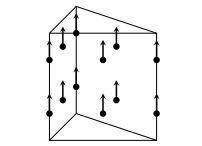


H^1: As in the 2D case, this is formed by taking the product of two H^1 elements.



$H(\text{curl})$: The element must again have nodes associated with edges of the reference product cell. There are two distinct ways of forming such an element, and in both cases a suitable Piola transform must be used to map functions from reference to physical space.

Taking the product of an H^1 2D element with an L^2 1D element produces a scalar-valued element with nodes on $(0, 1)$ edges. If we multiply this by the vector $(0, 0, 1)$, this results in an element whose tangential component is continuous on all edges and faces.



Alternatively, one may take the product of an $H(\text{div})$ or $H(\text{curl})$ 2D element with an H^1 1D element. This produces a vector-valued element with nodes on $(1, 0)$ edges. The product naturally takes values in $\mathbb{R}^2$, since the 2D element is vector-valued and the 1D element is scalar-valued. However, an $H(\text{curl})$ element in 3D must take values in $\mathbb{R}^3$. If the 2D element is in $H(\text{curl})$, it is enough to interpret the product as the first two components of a 3D vector. If the 2D element is in $H(\text{div})$, the 2D product must be *rotated* by 90 degrees before being transformed into a 3D vector.

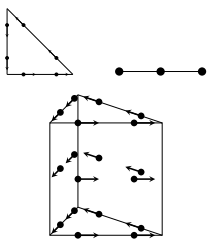
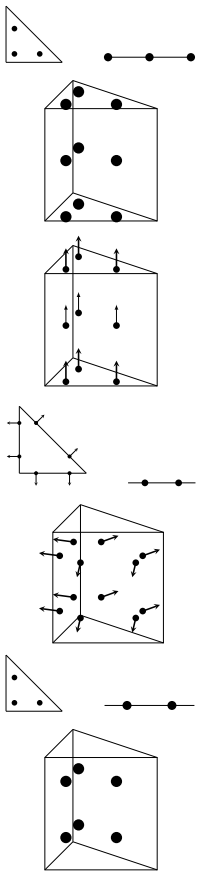


TABLE 2
Summary of 3D product elements.

Product (2D × 1D)	Components	Modifier	Result	Mapping
$H^1 \times H^1$	$f \times g$	(none)	fg	identity
$H^1 \times L^2$	$f \times g$	(none)	fg	identity
$H^1 \times L^2$	$f \times g$	$H(\mathrm{curl})$	$(0, 0, fg)$	covariant Piola
$H(\mathrm{curl}) \times H^1$	$(f_x, f_y) \times g$	(none)	$(f_x g, f_y g)^\dagger$	*
$H(\mathrm{curl}) \times H^1$	$(f_x, f_y) \times g$	$H(\mathrm{curl})$	$(f_x g, f_y g, 0)$	covariant Piola
$H(\mathrm{div}) \times H^1$	$(f_x, f_y) \times g$	(none)	$(f_x g, f_y g)^\dagger$	*
$H(\mathrm{div}) \times H^1$	$(f_x, f_y) \times g$	$H(\mathrm{curl})$	$(-f_y g, f_x g, 0)$	covariant Piola
$H(\mathrm{curl}) \times L^2$	$(f_x, f_y) \times g$	(none)	$(f_x g, f_y g)^\dagger$	*
$H(\mathrm{curl}) \times L^2$	$(f_x, f_y) \times g$	$H(\mathrm{div})$	$(f_y g, -f_x g, 0)$	contravariant Piola
$H(\mathrm{div}) \times L^2$	$(f_x, f_y) \times g$	(none)	$(f_x g, f_y g)^\dagger$	*
$H(\mathrm{div}) \times L^2$	$(f_x, f_y) \times g$	$H(\mathrm{div})$	$(f_x g, f_y g, 0)$	contravariant Piola
$L^2 \times H^1$	$f \times g$	(none)	fg	identity
$L^2 \times H^1$	$f \times g$	$H(\mathrm{div})$	$(0, 0, fg)$	contravariant Piola
$L^2 \times L^2$	$f \times g$	(none)	fg	identity

The elements marked with † are of little practical use; they are 2-vector valued but are defined over 3D domains. No mapping has been given for these elements; the Piola transformations from a 3D cell require all three components to be defined.



$H(\mathrm{div})$: The element must have nodes associated with facets of the reference product cell. As with $H(\mathrm{curl})$, there are two distinct ways of forming such an element, and suitable Piola transforms must again be used.

Taking the product of an L^2 2D element with an H^1 1D element gives a scalar-valued element with nodes on $(2, 0)$ facets. Multiplying this by $(0, 0, 1)$ produces an element whose normal component is continuous across all facets.

Taking the product of an $H(\mathrm{div})$ or $H(\mathrm{curl})$ 2D element with an L^2 1D element gives a vector-valued element with nodes on $(1, 1)$ facets. Again, the product naturally takes values in $\mathbb{R}^2$. If the 2D element is in $H(\mathrm{div})$, it is enough to interpret the product as the first two components of a 3D vector-valued element whose third component vanishes. If the 2D element is in $H(\mathrm{curl})$, the product must be rotated by 90 degrees before transforming.

L^2: As in the 2D case, both constituent elements must be in L^2.

2.4.5. Consequences for implementation. The previous subsections motivate the implementation of several mathematical operations on finite elements. We will need an operator that takes the product of two existing elements; we call this `TensorProductElement`. This will generate a new element whose reference cell is the product of the reference cells of the constituent elements, as described in subsection 2.4.1. It will also construct the product space of functions $P_A \otimes P_B$, as described in subsection 2.4.2, but with no extra manipulation (e.g., expanding into a vector-valued space). The basis for $P_A \otimes P_B$ is as defined in (11) and (12). The nodes are topologically associated with topological entities of the reference cell, as described in subsection 2.4.3.

To construct the more complicated vector-valued finite elements, we introduce additional operators `HCurl` and `HDiv` which form a vector-valued $H(\text{curl})$ or $H(\text{div})$ element from an existing `TensorProductElement`. This will modify the product space as described in subsection 2.4.4 by manipulating the existing product into a vector of the correct dimension (after rotation, if applicable), and setting an appropriate Piola transform. We will also need an operator that creates the sum of finite elements; this already exists in UFL under the name `EnrichedElement` and is represented by `+`.

2.5. Product finite elements within finite element exterior calculus. The work of Arnold, Falk, and Winther [7, 8] on finite element exterior calculus provides principles for obtaining stable mixed finite element discretizations on a domain consisting of simplicial cells: intervals, triangles, tetrahedra, and higher-dimensional analogues. In full generality, this involves de Rham complexes of polynomial-valued finite element differential forms linked by the exterior derivative operator. In 1D, 2D, and 3D, differential forms can be naturally identified with scalar and vector fields, while the exterior derivative can be interpreted as a standard differential operator such as grad, curl, or div. The vector-valued element spaces only have partial continuity between cells: they are in $H(\text{curl})$ or $H(\text{div})$, which have been discussed already. The element spaces themselves were, however, already well known in the existing finite element literature for their use in solving mixed formulations of the Poisson equation and problems of a similar nature.

Arnold, Boffi, and Bonizzoni [6] generalize finite element exterior calculus to cells which can be expressed as geometric products of simplices. They also describe a specific complex of finite element spaces on hexahedra (and, implicitly, quadrilaterals). When these differential forms are identified with scalar- and vector-valued functions, they correspond to the scalar-valued Q_r, its discontinuous counterpart DQ_r, and various well-known vector-valued spaces as introduced in Brezzi, Douglas, and Marini [14], Nédélec [32], and Nédélec [33]. Within finite element exterior calculus, there are element spaces which cannot be expressed as a tensor product of spaces on simplices (see, for example, Arnold and Awanou [5]), but we are not considering such spaces in this paper.

Finite element exterior calculus makes use of de Rham complexes of finite element spaces. In 1D, the complex takes the form

$$U_0 \xrightarrow{\frac{\mathrm{d}}{\mathrm{d}x}} U_1, \tag{13}$$

where $U_0 \subset H^1$ and $U_1 \subset L^2$. In 2D, there are two types of complexes, arising due to two possible identifications of differential 1-forms with vector fields:

$$U_0 \xrightarrow{\nabla^\perp} U_1 \xrightarrow{\nabla\cdot} U_2, \tag{14}$$

where $U_0 \subset H^1$, $U_1 \subset H(\mathrm{div})$, and $U_2 \subset L^2$, and

$$U_0 \xrightarrow{\nabla} U_1 \xrightarrow{\nabla^\perp \cdot} U_2, \tag{15}$$

where $U_0 \subset H^1$, $U_1 \subset H(\mathrm{curl})$, and $U_2 \subset L^2$. In 3D, the complex takes the form

$$U_0 \xrightarrow{\nabla} U_1 \xrightarrow{\nabla \times} U_2 \xrightarrow{\nabla \cdot} U_3, \tag{16}$$

where $U_0 \subset H^1$, $U_1 \subset H(\mathrm{curl})$, $U_2 \subset H(\mathrm{div})$, and $U_3 \subset L^2$.

Given an existing 2D complex (U_0, U_1, U_2) and a 1D complex (V_0, V_1), we can generate a product complex on the 3D product cell:

$$W_0 \xrightarrow{\nabla} W_1 \xrightarrow{\nabla \times} W_2 \xrightarrow{\nabla \cdot} W_3, \tag{17}$$

where

$$W_0 := U_0 \otimes V_0, \tag{18}$$

$$W_1 := \mathtt{HCurl}(U_0 \otimes V_1) \oplus \mathtt{HCurl}(U_1 \otimes V_0), \tag{19}$$

$$W_2 := \mathtt{HDiv}(U_1 \otimes V_1) \oplus \mathtt{HDiv}(U_2 \otimes V_0), \tag{20}$$

$$W_3 := U_2 \otimes V_1, \tag{21}$$

with $W_0 \subset H^1, W_1 \subset H(\mathrm{curl}), W_2 \subset H(\mathrm{div}), W_3 \subset L^2$ (compare with the complex given in (16)). The vector-valued spaces are direct sums of "product" spaces that have been modified by the `HCurl` or `HDiv` operator.

Similarly, taking the product of two 1D complexes produces a product complex on the 2D product cell in which the vector-valued space is in *either* $H(\mathrm{div})$ or $H(\mathrm{curl})$.

2.6. Product complexes using differential forms. This section summarizes Arnold, Boffi, and Bonizzoni [6] by restating the results of subsection 2.4 and subsection 2.5 in the language of differential forms, which can be considered a generalization of scalar and vector fields.

In 3D, 0-forms and 3-forms are identified with scalar fields, while 1-forms and 2-forms are identified with vector fields. In 2D, 0-forms and 2-forms are identified with scalar fields. 1-forms are identified with vector fields, but this can be done in two different ways since 1-forms and (n-1)-forms coincide. This results in two possible vector fields, which differ by a 90-degree rotation. In 1D, both 0-forms and 1-forms are conventionally identified with scalar fields.

Let $K_A \subset \mathbb{R}^n$, $K_B \subset \mathbb{R}^m$ be domains. Suppose we are given de Rham subcomplexes on K_A and K_B,

$$U_0 \xrightarrow{\mathrm{d}} U_1 \xrightarrow{\mathrm{d}} \cdots \xrightarrow{\mathrm{d}} U_n, \qquad V_0 \xrightarrow{\mathrm{d}} V_1 \xrightarrow{\mathrm{d}} \cdots \xrightarrow{\mathrm{d}} V_m, \tag{22}$$

where each U_k is a space of (polynomial) differential k-forms on K_A and each V_k is a space of differential k-forms on K_B. The product of these complexes is a de Rham subcomplex on $K_A \times K_B$:

$$(U \otimes V)_0 \xrightarrow{\mathrm{d}} (U \otimes V)_1 \xrightarrow{\mathrm{d}} \cdots \xrightarrow{\mathrm{d}} (U \otimes V)_{n+m}, \tag{23}$$

where, for $k = 0, 1, \ldots, n + m$,

$$(U \otimes V)_k := \bigoplus_{i+j=k} (U_i \otimes V_j). \tag{24}$$

Note that $(U \otimes V)_k$ is a space of (polynomial) k-forms on $K_A \otimes K_B$ and can hence be interpreted as a scalar or vector field in two or three spatial dimensions. It can be easily verified that the definitions in (23) and (24) gives rise to (17) in 3D, for example. The discussion in subsection 2.4.2 and subsection 2.4.4 on the product of function spaces can be summarized by the definition of $\otimes$ on the right-hand side of (24), along with the definition of the standard wedge product of differential forms. It is clear that much of the apparent complexity of the `HDiv` and `HCurl` operators introduced in subsection 2.4 arises from working with scalars and vectors rather than introducing differential forms!

3. Implementation. The symbolic operations on finite elements, derived in the previous section, have been implemented within Firedrake [36, 35]. Firedrake is an *automated system for the portable solution of partial differential equations using the finite element method*. Firedrake has several dependencies. Some of these are components of the FEniCS Project [27]:

FIAT FInite element Automatic Tabulator [22, 23], for the construction and tabulation of finite element basis functions.

UFL Unified Form Language [4, 2], a domain-specific language for the specification of finite element variational forms.

Firedrake also relies on PyOP2 [37] and COFFEE [30].

The changes required to effect the generation of product elements were largely confined to FIAT and UFL, while support for integration over product cells is included in Firedrake's form compiler. We begin this section with more detailed expositions on FIAT and UFL. We discuss the implementation of product finite elements in subsection 3.3. We talk about the resulting algebraic structure in subsection 3.4. We finish by discussing the new integration regions in subsection 3.5.

3.1. FIAT. This component is responsible for computing finite element basis functions for a wide range of finite element families. To do this, it works with an abstraction based on Ciarlet's definition of a finite element, as given in subsection 2.1. The reference cell K is defined using a set of vertices, with higher-dimensional geometrical objects defined as sets of vertices. The polynomial space P is defined implicitly through a *prime* basis: typically an orthonormal set of polynomials, such as (on triangles) a Dubiner basis, which can be stably evaluated to high polynomial order. The set of nodes N is also defined; this implies the existence of a *nodal* basis for P, as explained previously.

The nodal basis, which is important in calculations, can be expressed as linear combinations of prime basis functions. This is done automatically by FIAT; details are given in [22]. The main method of interacting with FIAT is by requesting the tabulated values of the nodal basis functions at a set of points inside K—typically a set of quadrature points. FIAT also stores the geometric decomposition of nodes relative to the topological entities of K.

3.2. UFL. This component is a domain-specific language, embedded in Python, for representing weak formulations of PDEs. It is centered around expressing multilinear forms: maps from the product of some set of function spaces $\{V_j\}_{j=1}^{\rho}$ into the real numbers which are linear in each argument, where ρ is 0, 1, or 2. Additionally, the form may be parameterized over one or more *coefficient functions* and is not necessarily linear in these. The form may include derivatives of functions, and the language has extensive support for matrix algebra operations.

We can assume that the function spaces are finite element spaces; in UFL, these

are represented by the `FiniteElement` class. This requires three pieces of information: the element family, the geometric cell, and the polynomial degree. A limited amount of symbolic manipulation on `FiniteElement` objects could already be done: the UFL `EnrichedElement` class is used to represent the $\oplus$ operator discussed in subsection 2.2.

3.3. Implementation of product finite elements. To implement product finite elements, additions to UFL and FIAT were required. The UFL changes are purely symbolic and allow the new elements to be represented. The FIAT changes allow the new elements (and derivatives thereof) to be numerically tabulated at specified points in the reference cell.

As discussed in subsection 2.4.5, we implemented several new element classes in UFL. The existing UFL `FiniteElement` classes have two essential properties: the `degree` and the `value_shape`. The `degree` is the maximal degree of any polynomial basis function; this allows determination of an appropriate quadrature rule. The `value_shape` represents whether the element is scalar-valued or vector-valued and, if applicable, the dimension of the vector in *physical* space. This allows suitable code to be generated when doing vector and tensor operations.

For `TensorProductElement`s, we define the `degree` to be a tuple; the basis functions are products of polynomials in distinct sets of variables. It is therefore advantageous to store the polynomial degrees separately for later use with a product quadrature rule. The `value_shape` is defined according to the definition in subsection 2.4.2 for the product of functions. For `HCurl` and `HDiv` elements, the `degree` is identical to the `degree` of the underlying `TensorProductElement`. The `value_shape` needs to be modified: in physical space, these vector-valued elements have dimension equal to the dimension of the physical space.

The secondary role of FIAT is to store a representation of the geometric decomposition of nodes. For product elements, the generation of this was described in subsection 2.4.3. The primary role is to tabulate finite element basis functions, and derivatives thereof, at specified points in the reference cell. The `tabulate` method of a FIAT finite element takes two arguments: the maximal `order` of derivatives to tabulate, and the set of `points`.

Let $\Phi_{i,j}(x,y,z) := \Phi_i^{(A)}(x,y)\Phi_j^{(B)}(z)$ be some product element basis function; we will assume that this is scalar-valued to ease the exposition. Suppose we need to tabulate the x-derivative of this at some specified point (x_0,y_0,z_0). Clearly

$$\frac{\partial \Phi_{i,j}}{\partial x}(x_0,y_0,z_0) = \frac{\partial \Phi_i^{(A)}}{\partial x}(x_0,y_0)\Phi_j^{(B)}(z_0). \tag{25}$$

In other words, the value can be obtained from tabulating (derivatives of) basis functions of the constituent elements at appropriate points. It is clear that this extends to other combinations of derivatives, as well as to components of vector-valued basis functions. Further modifications to the tabulation for curl- or div-conforming vector elements are relatively simple, as detailed in subsection 2.4.4.

3.4. Algebraic structure. The extensions described in subsection 3.3 enable sophisticated manipulation of finite elements within UFL. For example, consider the following complex on triangles, highlighted by Cotter and Shipton [19] as being relevant for numerical weather prediction:

$$\mathrm{P}_2 \oplus \mathrm{B}_3 \xrightarrow{\nabla^\perp} \mathrm{BDFM}_2 \xrightarrow{\nabla\cdot} \mathrm{DP}_1. \tag{26}$$

Here, $\mathrm{P}_2 \oplus \mathrm{B}_3$ denotes the space of quadratic polynomials enriched by a cubic "bubble" function, BDFM_2 represents a member of the vector-valued Brezzi–Douglas–Fortin–

LISTING 1
Construction of a complicated product complex in UFL.

```
U0_0 = FiniteElement("P", triangle, 2)
U0_1 = FiniteElement("B", triangle, 3)
U0 = EnrichedElement(U0_0, U0_1)
U1 = FiniteElement("BDFM", triangle, 2)
U2 = FiniteElement("DP", triangle, 1)

V0 = FiniteElement("P", interval, 1)
V1 = FiniteElement("DP", interval, 0)

W0 = TensorProductElement(U0, V0)
W1_h = TensorProductElement(U1, V0)
W1_v = TensorProductElement(U0, V1)
W1 = EnrichedElement(HCurl(W1_h), HCurl(W1_v))
W2_h = TensorProductElement(U1, V1)
W2_v = TensorProductElement(U2, V0)
W2 = EnrichedElement(HDiv(W2_h), HDiv(W2_v))
W3 = TensorProductElement(U2, V1)
```

Marini element family [15] in $H(\mathrm{div})$, and DP_1 represents the space of discontinuous, piecewise-linear functions. Suppose we wish to take the product of this with some complex on intervals, such as

$$\mathrm{P}_2 \xrightarrow{\frac{\mathrm{d}}{\mathrm{d}x}} \mathrm{DP}_1. \tag{27}$$

This generates a complex on triangular prisms:

$$W_0 \xrightarrow{\nabla} W_1 \xrightarrow{\nabla\times} W_2 \xrightarrow{\nabla\cdot} W_3, \tag{28}$$

where

$$W_0 := (\mathrm{P}_2^{\triangle} \oplus \mathrm{B}_3^{\triangle}) \otimes \mathrm{P}_2, \tag{29}$$

$$W_1 := \mathtt{HCurl}((\mathrm{P}_2^{\triangle} \oplus \mathrm{B}_3^{\triangle}) \otimes \mathrm{DP}_1) \oplus \mathtt{HCurl}(\mathrm{BDFM}_2^{\triangle} \otimes \mathrm{P}_2), \tag{30}$$

$$W_2 := \mathtt{HDiv}(\mathrm{BDFM}_2^{\triangle} \otimes \mathrm{DP}_1) \oplus \mathtt{HDiv}(\mathrm{DP}_1^{\triangle} \otimes \mathrm{P}_2), \tag{31}$$

$$W_3 := \mathrm{DP}_1^{\triangle} \otimes \mathrm{DP}_1; \tag{32}$$

we have marked the elements on triangles by $^{\triangle}$ for clarity. Following our extensions to UFL, the product complex may be constructed as shown in Listing 1. Some of these elements are used in the example in subsection 4.2.

3.5. Support for new integration regions. On simplicial meshes, Firedrake supports three types of integrals: integrals over cells, integrals over exterior facets, and integrals over interior facets. Integrals over exterior facets are typically used to apply boundary conditions weakly, while integrals over interior facets are used to couple neighboring cells when discontinuous function spaces are present. The implementation of the different types of integral is quite elegant: the only difference between integrating a function over the interior of the cell and over a single facet is the choice of quadrature points and quadrature weights. Note that Firedrake assumes that the mesh is conforming; hanging nodes are not currently supported.

On product cells, all entities can be considered as a product of entities on the constituent cells. We can therefore construct product quadrature rules, making use

of existing quadrature rules for constituent cells and facets thereof. In addition, we split the facet integrals into separate integrals over "vertical" and "horizontal" facets. This is natural when executing a computational kernel over an extruded unstructured mesh and may be useful in geophysical contexts where horizontal and vertical motions may be treated differently.

4. Numerical examples. In this section, we give several examples to demonstrate the correctness of our implementation. Quantitative analysis is performed where possible, e.g., demonstration of convergence to a known solution at expected order with increasing mesh resolution. Tests are performed in both two and three spatial dimensions. We make use of Firedrake's `ExtrudedMesh` functionality. In 2D, the cells are quadrilaterals, usually squares. In 3D, we use triangular prisms, though we can also build elements on hexahedra.

When referring to standard finite element spaces, we follow the convention in which the number refers to the degree of the minimal complete polynomial space containing the element, not the maximal complete polynomial space contained by the element. Thus, an element containing some, but not all, linear polynomials is numbered 1 rather than 0. This is the convention used by UFL and is also justified from the perspective of finite element exterior calculus.

4.1. Vector Laplacian (3D). We seek a solution to

$$-\nabla(\nabla\cdot\vec{u}) + \nabla\times(\nabla\times\vec{u}) = \vec{f} \tag{33}$$

in a domain Ω, with boundary conditions

$$\vec{u}\cdot\vec{n} = 0, \tag{34}$$

$$(\nabla\times\vec{u})\times\vec{n} = 0 \tag{35}$$

on $\partial\Omega$, where $\vec{n}$ is the outward normal. A naïve discretization can lead to spurious solutions, especially on nonconvex domains, but an accurate discretization can be obtained by introducing an auxiliary variable (see, for example, Arnold, Falk, and Winther [8]):

$$\sigma = -\nabla\cdot\vec{u}, \tag{36}$$

$$\nabla\sigma + \nabla\times(\nabla\times\vec{u}) = \vec{f}. \tag{37}$$

Let $V_0 \subset H^1$, $V_1 \subset H(\mathrm{curl})$ be finite element spaces. A suitable weak formulation is as follows: find $\sigma \in V_0$, $\vec{u} \in V_1$ such that

$$\langle\tau,\sigma\rangle - \langle\nabla\tau,\vec{u}\rangle = 0, \tag{38}$$

$$\langle\vec{v},\nabla\sigma\rangle + \langle\nabla\times\vec{v},\nabla\times\vec{u}\rangle = \langle\vec{v},\vec{f}\rangle \tag{39}$$

for all $\tau \in V_0$, $\vec{v} \in V_1$, where we have used angled brackets to denote the standard L^2 inner product. The boundary conditions have been implicitly applied, in a weak sense, through neglecting the surface terms when integrating by parts.

We take Ω to be the unit cube $[0,1]^3$. Let k, l, and m be arbitrary. Then

$$\vec{f} = \pi^2\begin{pmatrix}(k^2+l^2)\sin(k\pi x)\cos(l\pi y)\\(l^2+m^2)\sin(l\pi y)\cos(m\pi z)\\(k^2+m^2)\sin(m\pi z)\cos(k\pi x)\end{pmatrix} \tag{40}$$

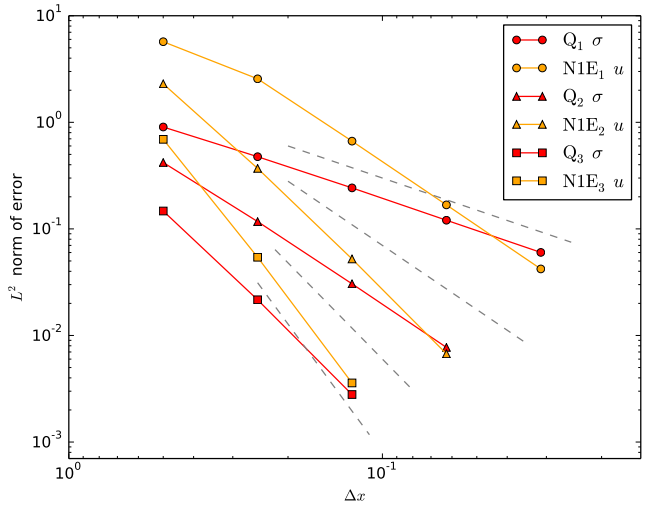


FIG. 1. *The L^2 error between the computed and "analytic" solution is plotted against Δx for the 3D problem described in subsection 4.1. The dotted lines are proportional to Δx^n, for n from 1 to 4, and are merely to aid comprehension.*

produces the solution

$$\vec{u} = \begin{pmatrix} \sin(k\pi x)\cos(l\pi y) \\ \sin(l\pi y)\cos(m\pi z) \\ \sin(m\pi z)\cos(k\pi x) \end{pmatrix}, \tag{41}$$

which satisfies the boundary conditions.

To discretize this problem, we subdivide Ω into triangular prisms whose base is a right-angled triangle with short sides of length Δx and whose height is Δx. We use the Q_r prism element for the H^1 space, and the degree-r Nédélec prism element of the first kind for the $H(\text{curl})$ space, for r from 1 to 3. We take k, l, and m to be 1, 2, and 3, respectively. We approximate $\vec{f}$ by interpolating the analytic expression onto a vector-valued function in Q_{r+1}. The L^2 errors between the calculated and "analytic" solutions for varying Δx are plotted in Figure 1. This is done for both $\vec{u}$ and σ; the so-called analytic solutions are approximations which are formed by interpolating the genuine analytic solution onto nodes of Q_{r+1}.

4.2. Gravity wave (3D). A simple model of atmospheric flow is given by

$$\frac{\partial \vec{u}}{\partial t} = -\nabla p + b\hat{z}, \quad \frac{\partial b}{\partial t} = -N^2 \vec{u}\cdot\hat{z}, \quad \frac{\partial p}{\partial t} = -c^2 \nabla\cdot\vec{u}, \tag{42}$$

along with the boundary condition $\vec{u}\cdot\vec{n} = 0$, where $\vec{n}$ is a unit normal vector. The prognostic variables are the velocity, $\vec{u}$, the pressure perturbation, p, and the buoyancy perturbation, b. The scalars N and c are (dimensional) constants, while $\hat{z}$ represents a unit vector opposite to the direction of gravity. These equations are a reduction of, for example, (17)–(21) from Skamarock and Klemp [45], in which we have neglected the constant background velocity and the Coriolis term and have rescaled θ by θ_0/g to produce b.

Given some 3D product complex as in (17), we seek a solution with $\vec{u} \in W_2^0$, $b \in W_2^v$, and $p \in W_3$. W_2^0 is the subspace of W_2 whose normal component vanishes on

the boundary of the domain. W_2^v denotes the "vertical" part of W_2: if we write W_2 as a sum of two product elements $\mathtt{HDiv}(U_1 \otimes V_1)$ and $\mathtt{HDiv}(U_2 \otimes V_0)$, then W_2^v is the *scalar-valued* product $U_2 \otimes V_0$, as constructed in Listing 1. This combination of finite element spaces for $\vec{u}$ and b is analogous to the Charney–Phillips staggering of variables in the vertical direction [17].

A semidiscrete form of (42) is the following: find $\vec{u} \in W_2^0$, $b \in W_2^v$, $p \in W_3$ such that for all $\vec{w} \in W_2^0$, $\gamma \in W_2^v$, $\phi \in W_3$

$$\left\langle \vec{w}, \frac{\partial \vec{u}}{\partial t} \right\rangle - \langle \nabla \cdot \vec{w}, p \rangle - \langle \vec{w}, b\hat{z} \rangle = 0, \tag{43}$$

$$\left\langle \gamma, \frac{\partial b}{\partial t} \right\rangle + N^2 \langle \gamma, \vec{u} \cdot \hat{z} \rangle = 0, \tag{44}$$

$$\left\langle \phi, \frac{\partial p}{\partial t} \right\rangle + c^2 \langle \phi, \nabla \cdot \vec{u} \rangle = 0. \tag{45}$$

It can be easily verified that the original equations, (42), together with the boundary condition lead to conservation of the energy perturbation

$$\int_\Omega \frac{1}{2}|\vec{u}|^2 + \frac{1}{2N^2}b^2 + \frac{1}{2c^2}p^2 \,\mathrm{d}x. \tag{46}$$

The three terms can be interpreted as kinetic energy (KE), potential energy (PE), and internal energy (IE), respectively. The semidiscretization given in (43)–(45) also conserves this energy. If we discretize in time using the implicit midpoint rule, which preserves quadratic invariants [25], then the fully discrete system will conserve energy as well.

We take the domain to be a spherical shell centered at the origin. Its inner radius, a, is approximately 6371km, and its thickness, H, is 10km. The domain is divided into triangular prism cells with side-lengths of the order of 1000km and height 1km. We take $N = 10^{-2}\mathrm{s}^{-1}$ and $c = 300\mathrm{ms}^{-1}$. The simulation starts at rest with a buoyancy perturbation and a vertically balancing pressure field given by

$$b = \frac{\sin(\pi(|\vec{x}| - a)/H)}{1 + z^2/L^2}, \qquad p = -\frac{H}{\pi}\frac{\cos(\pi(|\vec{x}| - a)/H)}{1 + z^2/L^2}; \tag{47}$$

L is a horizontal length-scale, which we take to be 500km. We use a timestep of the 1920s and run for a total of 480,000s.

To discretize this problem, we use the product elements formed from the BDFM_2 complex on triangles and the P_2–DP_1 complex on intervals; these were constructed in subsection 3.4. The initial conditions are interpolated into the buoyancy and pressure fields. The energy is calculated at every time step; the results are plotted in Figure 2. The total energy is conserved to roughly one part in 1.4×10^8, which is comparable to the linear solver tolerances.

4.3. DG advection (2D). The advection of a scalar field q by a known divergence-free velocity field $\vec{u_0}$ can be described by the equation

$$\frac{\partial q}{\partial t} + \nabla \cdot (\vec{u_0} q) = 0. \tag{48}$$

If q is in a discontinuous function space, V, a suitable weak formulation is

$$\left\langle \phi, \frac{\partial q}{\partial t} \right\rangle = \langle \nabla\phi, q\vec{u_0} \rangle - \int_{\Gamma_{\mathrm{ext}}} \phi \widetilde{q} \vec{u_0} \cdot \vec{n} \,\mathrm{d}s - \int_{\Gamma_{\mathrm{int}}} [\![\phi]\!] \widetilde{q} \vec{u_0} \cdot \vec{n} \,\mathrm{d}S \tag{49}$$

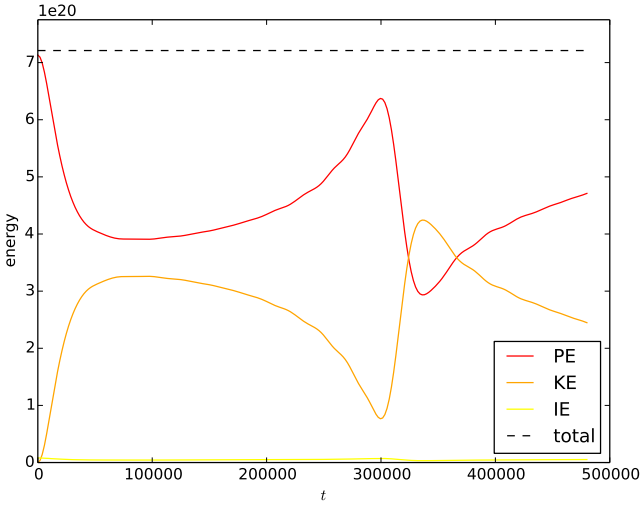


FIG. 2. *Evolution of energy for the simulation described in subsection* 4.2. *The components are the potential energy, PE, the kinetic energy, KE, and the internal energy, IE. The choice of spatial and temporal discretizations leads to exact conservation of total energy up to solver tolerances; this is indeed observed. The event at approximately* $t = 320{,}000s$ *corresponds to the zonally symmetric gravity wave reaching the poles of the spherical domain.*

for all $\phi \in V$, where the integrals on the right-hand side are over *exterior* and *interior* mesh facets, with $\mathrm{d}s$ and $\mathrm{d}S$ appropriate integration measures. $\vec{n}$ is the appropriately oriented normal vector, and $\widetilde{q}$ represents the *upwind* value of q, while $[\![\phi]\!]$ represents the jump in ϕ. We assume that, on parts of the boundary corresponding to inflow, $\widetilde{q} = 0$. This example will therefore demonstrate the ability to integrate over interior and exterior mesh facets.

We discretize (49) in time using the third-order three-stage strong-stability-preserving Runge–Kutta scheme given in [44]. We take Ω to be the unit square $[0,1]^2$. Our initial condition will be a cosine hill

$$q = \begin{cases} \frac{1}{2}\left(1 + \cos\left(\pi \frac{|\vec{x} - \vec{x_0}|}{r_0}\right)\right), & |\vec{x} - \vec{x_0}| < r_0, \\ 0 & \text{otherwise}, \end{cases} \tag{50}$$

with radius $r_0 = 0.15$, centered at $\vec{x_0} = (0.25, 0.5)$. The prescribed velocity field is

$$\vec{u_0}(\vec{x}, t) = \cos\left(\frac{\pi t}{T}\right) \begin{pmatrix} \sin(\pi x)^2 \sin(2\pi y) \\ -\sin(\pi y)^2 \sin(2\pi x) \end{pmatrix}, \tag{51}$$

as in LeVeque [26]. This gives a reversing, swirling flow field which vanishes on the boundaries of Ω. The initial condition should be recovered at $t = T$. Following [26], we take $T = \frac{3}{2}$.

To discretize this problem, we subdivide Ω into squares with side-length Δx. We use DQ_r for the discontinuous function space, for r from 0 to 2, which are products of 1D discontinuous elements. We initialize q by interpolating the expression given in (50) into the appropriate space. We approximate $\vec{u_0}$ by interpolating the expression given in (51) onto a vector-valued function in Q_2. The L^2 errors between the initial and final q fields for varying Δx are plotted in Figure 3.

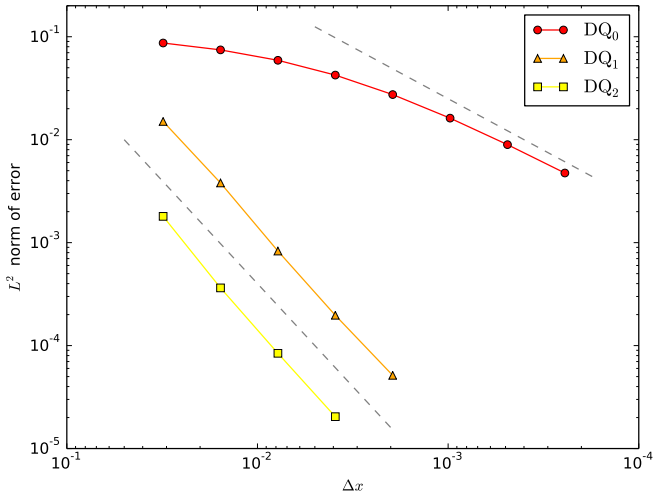


FIG. 3. *The L^2 error between the computed and "analytic" solution is plotted against Δx for the problem described in subsection* 4.3. *The dotted lines are proportional to Δx and Δx^2 and are merely to aid comprehension. The DQ_0 simulations converge at first order for sufficiently small values of Δx. The DQ_1 simulations converge at second order, as expected. The cosine bell initial condition has a discontinuous second derivative, which inhibits the DQ_2 simulations from exceeding a second-order rate of convergence.*

5. Limitations and extensions. There are several limitations of the current implementation, which leaves scope for future work. The most obvious is that the quadrature calculations are relatively inefficient, particularly at high order. The product structure of the basis functions can be exploited to generate a more efficient implementation of numerical quadrature. This can be done using the *sum-factorization* method, which lifts invariant terms out of the innermost loop. In the very simplest cases, direct factorization of the integral may be possible. Such operations could have been implemented within Firedrake's form compiler. However, this would mask the underlying issue—that FIAT, which is supposed to be wholly responsible for producing the finite elements, has no way to communicate any underlying basis function structure. Work is underway on a more sophisticated layer of software that returns an *algorithm* for performing a given operation on a finite element, rather than merely an array of tabulated basis functions.

Firedrake has recently gained full support for nonaffine coordinate transformations. In the *previous* version of the form compiler, the Jacobian of the coordinate mapping was assumed to be constant across each cell. This is satisfactory for simplices, since the physical and reference cells can always be linked by an affine transformation. However, this statement does not hold for quadrilateral, triangular prism, or hexahedral cells. Firedrake now evaluates the Jacobian at quadrature points. This functionality is also necessary for accurate calculations on curvilinear cells, in which the coordinate transformation is quadratic or higher-order. This allows, for example, more faithful representations of a sphere or spherical shell, extending the work done in [41].

6. Conclusion. This paper presented extensions to the automated code generation pipeline of Firedrake to facilitate the use of finite element spaces on nonsimplex

TABLE 3

Examples of the construction of standard finite element spaces. In the left-hand column, we use the notation of the Periodic Table of the Finite Elements [9] *where possible.*

Element	Cell	Construction*
Q_r (also written $\mathrm{Q}_{r,r}$)	quadrilateral	$\mathrm{P}_r \otimes \mathrm{P}_r$
RTCE_r, Raviart–Thomas "edge" element†	quadrilateral	$\mathtt{HCurl}(\mathrm{P}_r \otimes \mathrm{DP}_{r-1}) \oplus \mathtt{HCurl}(\mathrm{DP}_{r-1} \otimes \mathrm{P}_r)$
Nédélec "edge" element of the second kind‡	quadrilateral	$\mathtt{HCurl}(\mathrm{P}_r \otimes \mathrm{DP}_r) \oplus \mathtt{HCurl}(\mathrm{DP}_r \otimes \mathrm{P}_r)$
RTCF_r, Raviart–Thomas "face" element [39]	quadrilateral	$\mathtt{HDiv}(\mathrm{P}_r \otimes \mathrm{DP}_{r-1}) \oplus \mathtt{HDiv}(\mathrm{DP}_{r-1} \otimes \mathrm{P}_r)$
Nédélec "face" element of the second kind‡	quadrilateral	$\mathtt{HDiv}(\mathrm{P}_r \otimes \mathrm{DP}_r) \oplus \mathtt{HDiv}(\mathrm{DP}_r \otimes \mathrm{P}_r)$
DQ_r (discontinuous Q_r)	quadrilateral	$\mathrm{DP}_r \otimes \mathrm{DP}_r$
$\mathrm{P}_{r,r}{}^{\dagger\dagger}$	triangular prism	$\mathrm{P}_r^\triangle \otimes \mathrm{P}_r$
Nédélec "edge" element of the first kind‡‡	triangular prism	$\mathtt{HCurl}(\mathrm{P}_r^\triangle \otimes \mathrm{DP}_{r-1}) \oplus \mathtt{HCurl}(\mathrm{RTE}_r^\triangle \otimes \mathrm{P}_r)$
Nédélec "edge" element of the second kind [33]	triangular prism	$\mathtt{HCurl}(\mathrm{P}_r^\triangle \otimes \mathrm{DP}_r) \oplus \mathtt{HCurl}(\mathrm{BDME}_r^\triangle \otimes \mathrm{P}_r)$
Nédélec "face" element of the first kind‡‡	triangular prism	$\mathtt{HDiv}(\mathrm{RTF}_r^\triangle \otimes \mathrm{DP}_{r-1}) \oplus \mathtt{HDiv}(\mathrm{DP}_{r-1}^\triangle \otimes \mathrm{P}_r)$
Nédélec "face" element of the second kind [33]	triangular prism	$\mathtt{HDiv}(\mathrm{BDMF}_r^\triangle \otimes \mathrm{DP}_r) \oplus \mathtt{HDiv}(\mathrm{DP}_r^\triangle \otimes \mathrm{P}_r)$
$\mathrm{DP}_{r,r}$	triangular prism	$\mathrm{DP}_r^\triangle \otimes \mathrm{DP}_r$
Q_r (also written $\mathrm{Q}_{r,r,r}$)	hexahedra	$\mathrm{Q}_r^\square \otimes \mathrm{P}_r$
NCE_r, Nédélec "edge" element of the first kind [32]	hexahedra	$\mathtt{HCurl}(\mathrm{Q}_r^\square \otimes \mathrm{DP}_{r-1}) \oplus \mathtt{HCurl}(\mathrm{RTCE}_r^\square \otimes \mathrm{P}_r)$
Nédélec "edge" element of the second kind [33]	hexahedra	$\mathtt{HCurl}(\mathrm{Q}_r^\square \otimes \mathrm{DP}_r) \oplus \mathtt{HCurl}(\mathrm{N2CE}_r^\square \otimes \mathrm{P}_r)$
NCF_r, Nédélec "face" element of the first kind [32]	hexahedra	$\mathtt{HDiv}(\mathrm{RTCF}_r^\square \otimes \mathrm{DP}_{r-1}) \oplus \mathtt{HDiv}(\mathrm{DQ}_{r-1}^\square \otimes \mathrm{P}_r)$
Nédélec "face" element of the second kind [33]	hexahedra	$\mathtt{HDiv}(\mathrm{N2CF}_r^\square \otimes \mathrm{DP}_r) \oplus \mathtt{HDiv}(\mathrm{DQ}_r^\square \otimes \mathrm{P}_r)$
DQ_r	hexahedra	$\mathrm{DQ}_r^\square \otimes \mathrm{DP}_r$

†: This is a curl-conforming analogue of the usual Raviart–Thomas quadrilateral element [39].
‡: These are the quadrilateral reductions of the hexahedral Nédélec elements of the second kind [33].
††: This denotes the element with polynomial degree r in the first two variables, and polynomial degree r in the third variable separately.
‡‡: These are the prism equivalents of the tetrahedral and hexahedral Nédélec elements [32].
*: RTE and RTF refer to the Raviart–Thomas edge and face elements on triangles. BDME and BDMF refer to the Brezzi–Douglas–Marini [14] edge and face elements on triangles. N2CE and N2CF refer to the Nédélec elements of the second kind that we construct on quadrilaterals.

cells in 2D and 3D. A wide range of finite elements can be constructed, including, but not limited to, those listed in Table 3. The examples made extensive use of the recently added extruded mesh functionality in Firedrake; a related paper detailing the implementation of extruded meshes has been submitted [13].

All numerical experiments given in this paper were performed with the following versions of software, which we have archived on Zenodo: Firedrake [31], PyOP2 [38], TSFC [21], COFFEE [29], UFL [3], FIAT [40], PETSc [46], and PETSc4py [20]. The code for the numerical experiments can be found in the supplement to the paper (M102116_01.zip [local/web 7.73KB]).

REFERENCES

[1] MFEM Team, *MFEM: Modular Finite Element Methods*, http://mfem.org.

[2] M. S. Alnæs, *UFL: A finite element form language*, in Automated Solution of Differential Equations by the Finite Element Method, Lect. Notes Comp. Sci. Eng. 84, Springer, New York, 2012, pp. 303–338, doi:10.1007/978-3-642-23099-8_17.

[3] M. S. Alnæs, A. Logg, A. T. T. McRae, G. N. Wells, M. E. Rognes, L. Mitchell, M. Homolya, K. B. Ølgaard, A. Bergersen, J. Ring, D. A. Ham, C. Richardson, K.-A. Mardal, J. Blechta, F. Rathgeber, G. Markall, C. J. Cotter, L. Li, M. Liertzer, M. Albert, J. Hake, and T. Airaksinen, *UFL: The Unified Form Language*, 2016, doi:10.5281/zenodo.46250.

[4] M. S. Alnæs, A. Logg, K. B. Ølgaard, M. E. Rognes, and G. N. Wells, *Unified form language: A domain-specific language for weak formulations of partial differential equations*, ACM Trans. Math. Software, 40 (2014), 9, doi:10.1145/2566630.

[5] D. N. Arnold and G. Awanou, *Finite element differential forms on cubical meshes*, Math. Comp., 83 (2014), pp. 1551–1570, doi:10.1090/S0025-5718-2013-02783-4.

[6] D. N. Arnold, D. Boffi, and F. Bonizzoni, *Finite element differential forms on curvilinear cubic meshes and their approximation properties*, Numer. Math., (2014), pp. 1–20, doi:10.1007/s00211-014-0631-3.

[7] D. N. Arnold, R. S. Falk, and R. Winther, *Finite element exterior calculus, homological techniques, and applications*, Acta Numer., 15 (2006), pp. 1–155, dx.doi.org/10.1017/S0962492906210018.

[8] D. N. Arnold, R. S. Falk, and R. Winther, *Finite element exterior calculus: From Hodge theory to numerical stability*, Bull. Amer. Math. Soc. (N.S.), 47 (2010), pp. 281–354, doi:10.1090/S0273-0979-10-01278-4.

[9] D. N. Arnold and A. Logg, *Periodic table of the finite elements*, SIAM News, (November) 2014; also available online from http://femtable.org.

[10] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, K. Rupp, B. F. Smith, and H. Zhang, *PETSc Users Manual*, Tech. report ANL-95/11 - Revision 3.5, Argonne National Laboratory, Lemont, IL, 2014, http://www.mcs.anl.gov/petsc.

[11] W. Bangerth, T. Heister, L. Heltai, G. Kanschat, M. Kronbichler, M. Maier, B. Turcksin, and T. D. Young, *The* `deal.II` *library, version* 8.2, Arch. Numer. Software, 3 (2015), doi:10.11588/ans.2015.100.18031.

[12] P. Bastian, F. Heimann, and S. Marnach, *Generic implementation of finite element methods in the Distributed and Unified Numerics Environment (DUNE)*, Kybernetika, 46 (2010), pp. 294–315.

[13] G.-T. Bercea, A. T. T. McRae, D. A. Ham, L. Mitchell, F. Rathgeber, L. Nardi, F. Luporini, and P. H. J. Kelly, *A Numbering Algorithm for Finite Element on Extruded Meshes which Avoids the Unstructured Mesh Penalty*, preprint, arxiv.org/abs/1604.05937 [cs.MS], 2016.

[14] F. Brezzi, J. Douglas, Jr., and L. D. Marini, *Two families of mixed finite elements for second order elliptic problems*, Numer. Math., 47 (1985), pp. 217–235, doi:10.1007/BF01389710.

[15] F. Brezzi and M. Fortin, *Mixed and Hybrid Finite Element Methods*, Springer Ser. Comput. Math. 15, Springer-Verlag, New York, 1991.

[16] C. Cantwell, D. Moxey, A. Comerford, A. Bolis, G. Rocco, G. Mengaldo, D. D. Grazia, S. Yakovlev, J.-E. Lombard, D. Ekelschot, B. Jordi, H. Xu, Y. Mohamied, C. Eskilsson, B. Nelson, P. Vos, C. Biotto, R. Kirby, and S. Sherwin, *Nektar++: An open-source spectral/*hp *element framework*, Comput. Phys. Comm., 192 (2015), pp. 205–219, doi:10.1016/j.cpc.2015.02.008.

[17] J. G. Charney and N. A. Phillips, *Numerical integration of the quasi-geostrophic equations for barotropic and simple baroclinic flows*, J. Meteorology, 10 (1953), pp. 71–99, doi:10.1175/1520-0469(1953)010%3C0071:NIOTQG%3E2.0.CO;2.

[18] P. G. Ciarlet, *The Finite Element Method for Elliptic Problems*, North–Holland, Amsterdam, 1978.

[19] C. J. Cotter and J. Shipton, *Mixed finite elements for numerical weather prediction*, J. Comput. Phys., 231 (2012), pp. 7076–7091, doi:10.1016/j.jcp.2012.05.020.

[20] L. Dalcin, L. Mitchell, J. Brown, P. E. Farrell, M. Lange, B. Smith, D. Karpeyev, N. O. Collier, M. Knepley, D. A. Ham, S. W. Funke, A. Ahmadia, T. Hisch, M. Homolya, J. C. Alastuey, A. N. Riseth, G. Wells, and J. Guyer, *PETSc4py: The Python Interface to PETSc*, Zenodo, 2016, doi:10.5281/zenodo.46222.

[21] M. Homolya and L. Mitchell, *TSFC: The Two Stage Form Compiler*, Zenodo, 2016, doi:10.5281/zenodo.46217.

[22] R. C. Kirby, *Algorithm* 839: *FIAT, a new paradigm for computing finite element basis functions*, ACM Trans. Math. Softw., 30 (2004), pp. 502–516, doi:10.1145/1039813.1039820.

[23] R. C. Kirby, *FIAT: Numerical construction of finite element basis functions*, in Automated Solution of Differential Equations by the Finite Element Method, Lect. Notes Comput. Sci. Eng. 84, Springer, New York, 2012, pp. 247–255, doi:10.1007/978-3-642-23099-8_13.

[24] R. C. Kirby and A. Logg, *A compiler for variational forms*, ACM Trans. Math. Softw., 32 (2006), pp. 417–444, doi:10.1145/1163641.1163644.

[25] B. Leimkuhler and S. Reich, *Simulating Hamiltonian Dynamics*, Cambridge University Press, Cambridge, UK, 2005, Ch. 12, doi:10.1017/CBO9780511614118.

[26] R. J. LeVeque, *High-resolution conservative algorithms for advection in incompressible flow*, SIAM J. Numer. Anal., 33 (1996), pp. 627–665, doi:10.1137/0733033.

[27] A. Logg, K.-A. Mardal, and G. N. Wells, *Automated Solution of Differential Equations by the Finite Element Method*, Lect. Notes Comput. Sci. Eng. 84, Springer, Heidelberg, 2012, doi:10.1007/978-3-642-23099-8.

[28] A. Logg, K. B. Ølgaard, M. E. Rognes, and G. N. Wells, *FFC: The FEniCS form compiler*, in Automated Solution of Differential Equations by the Finite Element Method, Lect. Notes Comput. Sci. Eng. 84, Springer, Heidelberg, 2012, pp. 227–238, doi:10.1007/978-3-642-23099-8_11.

[29] F. Luporini, L. Mitchell, M. Homolya, F. Rathgeber, D. A. Ham, M. Lange, G. Markall, and F. Russell, *COFFEE: A Compiler for Fast Expression Evaluation*, Zenodo, 2016, doi:10.5281/zenodo.46218.

[30] F. Luporini, A. L. Varbanescu, F. Rathgeber, G.-T. Bercea, J. Ramanujam, D. A. Ham, and P. H. J. Kelly, *Cross-loop optimization of arithmetic intensity for finite element local assembly*, ACM Trans. Architecture Code Optim., 11 (2015), pp. 57:1–57:25, doi:10.1145/2687415.

[31] L. Mitchell, F. Rathgeber, D. A. Ham, M. Homolya, A. T. T. McRae, G.-T. Bercea, M. Lange, C. J. Cotter, C. T. Jacobs, F. Luporini, S. W. Funke, A. Kalogirou, H. Büsing, T. Kärnä, H. Rittich, E. H. Mueller, S. Kramer, G. Markall, P. E. Farrell, A. N. Riseth, J. Chang, and G. McBain, *Firedrake: An Automated Finite Element System*, Zenodo, 2016, doi:10.5281/zenodo.46221.

[32] J. C. Nédélec, *Mixed finite elements in* $\mathbb{R}^3$, Numer. Math., 35 (1980), pp. 315–341, doi:10.1007/BF01396415.

[33] J. C. Nédélec, *A new family of mixed finite elements in* $\mathbb{R}^3$, Numer. Math., 50 (1986), pp. 57–81, doi:10.1007/BF01389668.

[34] K. B. Ølgaard and G. N. Wells, *Optimisations for quadrature representations of finite element tensors through automated code generation*, ACM Trans. Math. Softw., 37 (2010), pp. 8:1–8:23, doi:10.1145/1644001.1644009.

[35] F. Rathgeber, *Productive and Efficient Computational Science through Domain-Specific Abstractions*, Ph.D. thesis, Imperial College London, London, 2014.

[36] F. Rathgeber, D. A. Ham, L. Mitchell, M. Lange, F. Luporini, A. T. T. McRae, G.-T. Bercea, G. R. Markall, and P. H. J. Kelly, *Firedrake: Automating the finite element method by composing abstractions*, submitted.

[37] F. Rathgeber, G. R. Markall, L. Mitchell, N. Loriant, D. A. Ham, C. Bertolli, and P. H. Kelly, *PyOP2: A high-level framework for performance-portable simulations on unstructured meshes*, in SC Companion: High Performance Computing, Networking Storage and Analysis, IEEE Computer Society, Los Alamitos, CA, 2012, pp. 1116–1123, doi:10.1109/SC.Companion.2012.134.

[38] F. Rathgeber, L. Mitchell, F. Luporini, G. Markall, D. A. Ham, G.-T. Bercea, M. Homolya, A. T. T. McRae, H. Dearman, C. T. Jacobs, G. Boutsioukis, S. W. Funke, K. Sato, and F. Russell, *PyOP2: Framework for Performance-Portable Parallel Computations on Unstructured Meshes*, Zenodo, 2016, doi:10.5281/zenodo.46219.

[39] P. A. Raviart and J. M. Thomas, *A mixed finite element method for* 2*nd order elliptic problems*, in Mathematical Aspects of Finite Element Methods, Springer, New York, 1977, pp. 292–315, doi:10.1007/BFb0064470.

[40] M. E. Rognes, A. Logg, M. Homolya, D. A. Ham, N. Schlömer, J. Blechta, A. Bergersen, J. Ring, C. J. Cotter, L. Mitchell, G. Wells, F. Rathgeber, R. Kirby, L. Li, M. S. Alnæs, A. T. T. McRae, and M. Liertzer, *FIAT: The Finite Element Automated Tabulator*, Zenodo, 2016, doi:10.5281/zenodo.46220.

[41] M. E. Rognes, D. A. Ham, C. J. Cotter, and A. T. T. McRae, *Automating the solution of pdes on the sphere and other manifolds in FEniCS* 1.2, Geosci. Model Development, 6 (2013), pp. 2099–2119, doi:10.5194/gmd-6-2099-2013.

[42] J. Schöberl, *C++11 Implementation of Finite Elements in NGSolve*, preprint, Institute for Analysis and Scientific Computing, TU Wien, Vienna, Austria, 2014; available online at http://www.asc.tuwien.ac.at/preprint/2014/asc30x2014.pdf.

[43] J. Schöberl and S. Zaglmayr, *High order Nédélec elements with local complete sequence properties*, COMPEL: The International Journal for Computation and Mathematics in Electrical and Electronic Engineering, 24 (2005), pp. 374–384, doi:10.1108/03321640510586015.

[44] C.-W. Shu and S. Osher, *Efficient implementation of essentially non-oscillatory shock-capturing schemes*, J. Comput. Phys., 77 (1988), pp. 439–471, doi:10.1016/0021-9991(88)90177-5.

[45] W. C. Skamarock and J. B. Klemp, *Efficiency and accuracy of the Klemp-Wilhelmson time-splitting technique*, Monthly Weather Review, 122 (1994), pp. 2623–2630, http://dx.doi.org/10.1175/1520-0493(1994)122⟨2623:EAAOTK⟩2.0.CO;2.

[46] B. Smith, S. Balay, M. Knepley, J. Brown, L. C. McInnes, H. Zhang, P. Brune, J. Sarich, D. Karpeyev, L. Dalcin, S. Zampini, M. Adams, V. Minden, V. Eijkhout, V. S. Mahadevan, T. Isaac, K. Rupp, S. Han, M. Lange, D. Meiser, X. Zhou, B. Aagaard, D. May, T. Munson, E. M. Constantinescu, D. Ghosh, L. Mitchell, P. Sanan, and B. A. Bourdin, *PETSc: Portable, Extensible Toolkit for Scientific Computation*, Feb. 2016, doi:10.5281/zenodo.46181.